generalordinal

jaume cloquell capo

6 de febrero de 2019

Clasificación ordinal

Lectura del dataset

El primer paso es leer los datos, al ser un fichero **arff** usaremos la funcion read.arff de Rweka.

```
original <- read.arff("esl.arff")
#original <- read.arff("era.arff")
#original <- read.arff("lev.arff")
#original <- read.arff("swd.arff")
dt <- original
summary(dt)
```

```
##       in1             in2             in3             in4
##  Min.   :0.000   Min.   :0.000   Min.   :2.000   Min.   :2.000
##  1st Qu.:4.750   1st Qu.:4.000   1st Qu.:5.000   1st Qu.:5.000
##  Median :6.000   Median :5.000   Median :5.000   Median :6.000
##  Mean   :5.506   Mean   :5.084   Mean   :5.344   Mean   :5.611
##  3rd Qu.:6.000   3rd Qu.:6.000   3rd Qu.:6.000   3rd Qu.:6.000
##  Max.   :9.000   Max.   :9.000   Max.   :8.000   Max.   :8.000
##       out1
##  Min.   :1.00
##  1st Qu.:4.00
##  Median :5.00
##  Mean   :5.23
##  3rd Qu.:6.00
##  Max.   :9.00
```

```
head(dt)
```

```
##   in1 in2 in3 in4 out1
## 1   6   5   6   6    6
## 2   5   4   5   5    5
## 3   5   3   4   5    4
## 4   6   5   6   7    6
## 5   4   3   3   5    3
## 6   9   6   6   6    6
```

Descomposición del problema

Para poder aplicar los modelos múltiples de clasificación ordinal tendremos que descomponer el problema en el caso de nuestro problema con 9 clases, tendremos que crear 8 dataframes. La idea de no usar el 9º dataset es que al ir calculando las probabilidades en cascada, si llegamos al 8 y no hemos clasificado correctamente implicará que estamos ante un ejemplo de la última por eliminación.

```
clases=as.integer(unique(dt$out1))
sort(clases)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Seleccionaremos los índices de éstas:

```
indices<-which(dt$out1==clases[1])
indices
```

```
##   [1]   1   4   6  10  11  12  18  21  24  29  30  31  35  38  54  55  59
##  [18]  60  62  64  67  68  69  70  73  74  77  82  84  91  92  93  98 100
##  [35] 101 114 116 117 118 119 120 128 130 131 135 144 145 146 149 156 158
##  [52] 161 164 165 169 171 173 174 177 179 183 206 207 211 214 215 217 227
##  [69] 235 239 240 244 246 252 259 262 265 268 291 293 294 305 308 310 311
##  [86] 315 316 317 324 326 329 331 332 338 340 341 345 352 356 357 360 363
## [103] 365 366 368 369 374 375 379 383 385 386 387 390 391 396 400 406 407
## [120] 413 414 419 420 430 434 438 445 454 456 457 467 471 478 485 486
```

Guardamos la variable clase en un vector auxiliar

```
y = as.integer(dt$out1)
y
```

```
##   [1] 6 5 4 6 3 6 4 2 5 6 6 6 3 5 5 4 5 6 4 7 6 2 5 6 7 7 2 5 6 6 6 4 3 5 6
##  [36] 4 5 6 4 4 5 4 5 7 7 2 5 3 4 5 4 5 7 6 6 5 5 7 6 6 3 6 4 6 5 8 6 6 6 6
##  [71] 5 4 6 6 5 4 6 5 4 5 7 6 7 6 7 8 3 5 7 7 6 6 6 7 5 7 4 6 4 6 6 7 5 1 5
## [106] 8 4 9 7 7 5 8 4 6 5 6 6 6 6 6 7 4 1 5 5 3 4 6 3 6 6 7 3 4 6 3 4 5 5 4
## [141] 4 7 8 6 6 6 5 4 6 5 4 3 4 4 4 6 5 6 7 4 6 8 4 6 6 5 5 8 6 4 6 4 6 6 4
## [176] 5 6 4 6 8 5 8 6 7 4 5 5 3 7 4 5 4 4 5 5 7 8 2 5 2 5 3 3 8 7 6 6 4 8 9
## [211] 6 3 5 6 6 9 6 5 4 4 5 4 4 9 5 4 6 8 4 3 4 7 5 5 6 4 5 5 6 6 5 3 4 6 5
## [246] 6 2 3 5 4 5 6 2 4 7 4 4 5 6 4 7 6 5 4 6 5 5 6 7 7 2 3 5 5 4 4 3 3 3 7
## [281] 5 7 5 7 7 8 8 5 4 5 6 4 6 6 7 4 4 4 5 3 7 7 2 5 6 5 5 6 4 6 6 5 4 4 6
## [316] 6 6 3 8 4 4 5 3 6 5 6 5 4 6 7 6 6 5 5 5 3 5 6 5 6 6 5 7 5 6 8 7 4 7 5
## [351] 5 6 5 3 5 6 6 7 4 6 5 5 6 7 6 6 4 6 6 5 4 5 4 6 6 7 4 4 6 4 3 7 6 7 6
## [386] 6 6 7 5 6 6 5 4 7 3 6 4 3 4 6 5 4 5 4 4 6 6 7 5 8 4 4 6 6 5 4 5 4 6 6
## [421] 7 7 5 4 3 3 5 4 2 6 5 4 5 6 3 7 5 6 7 5 4 5 4 7 6 5 4 3 5 5 4 3 5 6 5
## [456] 6 6 7 5 5 7 3 7 5 4 7 6 4 8 5 6 4 4 4 4 3 2 6 5 7 5 7 7 3 6 6 7 5
```

Cambiamos los valores de estas clases a 0 y el resto a 1

```
y[indices]<-0
y = ifelse(y==0,0,1)
```

Con esto ya tenemos casi listo el primer data frame derivado, nos queda por juntar el resto del dataset con la nueva clase binaria. Más adelante se procederá a clasificar dicho conjunto de datos , por lo que es conveniente pasar la variable a factor

```
data1 = cbind(dt[,1:4],target1=as.factor(y))
sapply(data1,class)
```

```
##       in1       in2       in3       in4   target1
## "numeric" "numeric" "numeric" "numeric"  "factor"
```

```
head(data1)
```

```
##   in1 in2 in3 in4 target1
## 1   6   5   6   6       0
## 2   5   4   5   5       1
## 3   5   3   4   5       1
## 4   6   5   6   7       0
## 5   4   3   3   5       1
```

```
## 6   9   6   6   6       0
```

Rocederemos a repetir los pasos anteriores, teniendo en cuenta las clases ya convertidas en el paso anterior, por lo que las añadiremos al vector de índices previamente creado.

```
indices<-c(indices,which(dt$out1==clases[2]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data2 <- cbind(dt[,1:4],target2=as.factor(y))

indices<-c(indices,which(dt$out1==clases[3]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data3 <- cbind(dt[,1:4],target3=as.factor(y))

indices<-c(indices,which(dt$out1==clases[4]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data4 <- cbind(dt[,1:4],target4=as.factor(y))

indices<-c(indices,which(dt$out1==clases[5]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data5 <- cbind(dt[,1:4],target5=as.factor(y))

indices<-c(indices,which(dt$out1==clases[6]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data6 <- cbind(dt[,1:4],target6=as.factor(y))

indices<-c(indices,which(dt$out1==clases[7]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data7 <- cbind(dt[,1:4],target7=as.factor(y))

indices<-c(indices,which(dt$out1==clases[8]))
y = as.integer(dt$out1)
y[indices]=0
y = ifelse(y==0,0,1)
data8 <- cbind(dt[,1:4],target8=as.factor(y))
```

Clasificación

Creamos un modelo para cada uno de nuestros subproblemas binarios.

```
library(RWeka)
m1 <- J48(target1 ~ ., data = data1)
m2 <- J48(target2 ~ ., data = data2)
```

```
m3 <- J48(target3 ~ ., data = data3)
m4 <- J48(target4 ~ ., data = data4)
m5 <- J48(target5 ~ ., data = data5)
m6 <- J48(target6 ~ ., data = data6)
m7 <- J48(target7 ~ ., data = data7)
m8 <- J48(target8 ~ ., data = data8)
```

Podemos hacer un estudio más detallado de los modelos , haciendo uso de la siguiente función

```
eval_m1 <- evaluate_Weka_classifier(m1, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m2 <- evaluate_Weka_classifier(m2, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m3 <- evaluate_Weka_classifier(m3, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m4 <- evaluate_Weka_classifier(m4, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m5 <- evaluate_Weka_classifier(m5, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m6 <- evaluate_Weka_classifier(m6, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m7 <- evaluate_Weka_classifier(m7, numFolds = 10, complexity = FALSE, class = TRUE)
eval_m8 <- evaluate_Weka_classifier(m8, numFolds = 10, complexity = FALSE, class = TRUE)
```

```
eval_m1
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         412               84.4262 %
## Incorrectly Classified Instances        76               15.5738 %
## Kappa statistic                          0.6246
## Mean absolute error                      0.1987
## Root mean squared error                  0.3521
## Relative absolute error                 49.5942 %
## Root relative squared error             78.7041 %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,778    0,130    0,695      0,778    0,734      0,627    0,876     0,640     0
##                  0,870    0,222    0,911      0,870    0,890      0,627    0,876     0,946     1
## Weighted Avg.    0,844    0,197    0,851      0,844    0,847      0,627    0,876     0,861
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  105  30 |   a = 0
##   46 307 |   b = 1
```

```
eval_m2
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         420               86.0656 %
## Incorrectly Classified Instances        68               13.9344 %
## Kappa statistic                          0.7207
## Mean absolute error                      0.2169
```

```
## Root mean squared error                  0.3522
## Relative absolute error                 43.405  %
## Root relative squared error             70.473  %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,888    0,169    0,848      0,888    0,868      0,722    0,856     0,800     0
##                  0,831    0,112    0,876      0,831    0,853      0,722    0,856     0,827     1
## Weighted Avg.    0,861    0,141    0,861      0,861    0,860      0,722    0,856     0,813
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  223  28 |   a = 0
##   40 197 |   b = 1
```

```
eval_m3
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         422               86.4754 %
## Incorrectly Classified Instances        66               13.5246 %
## Kappa statistic                          0.6575
## Mean absolute error                      0.1982
## Root mean squared error                  0.3404
## Relative absolute error                 49.0332 %
## Root relative squared error             75.7399 %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,920    0,277    0,895      0,920    0,907      0,658    0,843     0,900     0
##                  0,723    0,080    0,780      0,723    0,750      0,658    0,843     0,673     1
## Weighted Avg.    0,865    0,222    0,862      0,865    0,863      0,658    0,843     0,836
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  323  28 |   a = 0
##   38  99 |   b = 1
```

```
eval_m4
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         448               91.8033 %
## Incorrectly Classified Instances        40                8.1967 %
## Kappa statistic                          0.7367
## Mean absolute error                      0.1391
```

```
## Root mean squared error                  0.2745
## Relative absolute error                 42.8988 %
## Root relative squared error             68.2536 %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,961    0,253    0,937      0,961    0,949      0,738    0,855     0,931     0
##                  0,747    0,039    0,831      0,747    0,787      0,738    0,855     0,712     1
## Weighted Avg.    0,918    0,209    0,916      0,918    0,916      0,738    0,855     0,886
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  374  15 |   a = 0
##   25  74 |   b = 1
```

```
eval_m5
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         453               92.8279 %
## Incorrectly Classified Instances        35                7.1721 %
## Kappa statistic                          0.7472
## Mean absolute error                      0.1147
## Root mean squared error                  0.2594
## Relative absolute error                 39.0053 %
## Root relative squared error             67.7623 %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,965    0,241    0,949      0,965    0,957      0,748    0,866     0,940     0
##                  0,759    0,035    0,825      0,759    0,790      0,748    0,866     0,659     1
## Weighted Avg.    0,928    0,205    0,927      0,928    0,927      0,748    0,866     0,890
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  387  14 |   a = 0
##   21  66 |   b = 1
```

```
eval_m6
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         471               96.5164 %
## Incorrectly Classified Instances        17                3.4836 %
## Kappa statistic                          0.5475
## Mean absolute error                      0.0591
```

```
## Root mean squared error                  0.1757
## Relative absolute error                 59.6297 %
## Root relative squared error             79.685  %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  0,994    0,560    0,970      0,994    0,982      0,573    0,719     0,968     0
##                  0,440    0,006    0,786      0,440    0,564      0,573    0,719     0,455     1
## Weighted Avg.    0,965    0,532    0,961      0,965    0,960      0,573    0,719     0,941
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  460   3 |   a = 0
##   14  11 |   b = 1
```

```
eval_m7
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         482               98.7705 %
## Incorrectly Classified Instances         6                1.2295 %
## Kappa statistic                          0
## Mean absolute error                      0.0243
## Root mean squared error                  0.1103
## Relative absolute error                 91.8506 %
## Root relative squared error             99.9806 %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  1,000    1,000    0,988      1,000    0,994      ?        0,299     0,982     0
##                  0,000    0,000    ?          0,000    ?          ?        0,299     0,012     1
## Weighted Avg.    0,988    0,988    ?          0,988    ?          ?        0,299     0,970
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  482   0 |   a = 0
##    6   0 |   b = 1
```

```
eval_m8
```

```
## === 10 Fold Cross Validation ===
##
## === Summary ===
##
## Correctly Classified Instances         484               99.1803 %
## Incorrectly Classified Instances         4                0.8197 %
## Kappa statistic                          0
## Mean absolute error                      0.0163
```

```
## Root mean squared error                  0.0903
## Relative absolute error                 88.128  %
## Root relative squared error             99.9704 %
## Total Number of Instances              488
##
## === Detailed Accuracy By Class ===
##
##                  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
##                  1,000    1,000    0,992      1,000    0,996      ?        0,198     0,986     0
##                  0,000    0,000    ?          0,000    ?          ?        0,198     0,008     1
## Weighted Avg.    0,992    0,992    ?          0,992    ?          ?        0,198     0,978
##
## === Confusion Matrix ===
##
##    a   b   <-- classified as
##  484   0 |   a = 0
##    4   0 |   b = 1
```

Necesitamos conocer las probabilidades generadas por nuestros modelos, para ello probaremos a predecir la instancia numero 108 de nuestro dataset, sabiendo de por si que pertenece a la clase 9. La clase con máxima probabilidad se asigna a la instancia. Es decir en este caso es prop9

```
pred1<-predict(m1,dt[108,1:4],type="probability")
prop1 <- 1 - pred1[1]
pred2<-predict(m2,dt[108,1:4],type="probability")
prop2 <- (1 - pred1[1] ) * pred2[1]
pred3<-predict(m3,dt[108,1:4],type="probability")
prop3 <- (1 - pred2[1] ) * pred3[1]
pred4<-predict(m4,dt[108,1:4],type="probability")
prop4 <- (1 - pred3[1] ) * pred4[1]
pred5<-predict(m5,dt[108,1:4],type="probability")
prop5 <- (1 - pred4[1] ) * pred5[1]
pred6<-predict(m6,dt[108,1:4],type="probability")
prop6 <- (1 - pred5[1] ) * pred6[1]
pred7<-predict(m7,dt[108,1:4],type="probability")
prop7 <- (1 - pred6[1] ) * pred7[1]
pred8<-predict(m8,dt[108,1:4],type="probability")
prop8 <- (1 - pred7[1] ) * pred8[1]
prop9 <-  pred8[1]
prop1
```

```
## [1] 0.8813559
```

```
prop2
```

```
## [1] 0.1279388
```

```
prop3
```

```
## [1] 0.1240895
```

```
prop4
```

```
## [1] 0.1240895
```

```
prop5
```

```
## [1] 0.1240895
```

```
prop6
```

```
## [1] 0.8141321
```

```
prop7
```

```
## [1] 0.04703357
```

```
prop8
```

```
## [1] 0.0121943
```

```
prop9
```

```
## [1] 0.9918033
```

Una vez tengamos este resultado, debemos agregar los resultados para obtener la clasificación de nuestro modelo para el conjunto de test. Esto lo haremos todo en una función que generalice el modelo y realice todo el calculo de manera que nos ofrezca como salida un vector con las clase predicha.

Generalización del proceso de modelos múltiples

```
rm(list=ls())
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 3.5.2
```

```
library(Matrix)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(RWeka)
library(data.table)
library(caTools)
```

```
##
## Attaching package: 'caTools'
```

```
## The following object is masked from 'package:RWeka':
##
##     LogitBoost
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following object is masked from 'package:xgboost':
##
##     slice
```

```
## The following objects are masked from 'package:stats':
##
```

```
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(randomForest)
```

```
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caTools)

esl <- read.arff("esl.arff")
era <- read.arff("era.arff")
lev <- read.arff("lev.arff")
swd <- read.arff("swd.arff")

#'@function createTraintAndTestPartition
#'@description General las particiones de test y train
#'
#'@param dataset Dataset original
#'@param SplitRatio Porcentaje de valores en train y test
#'@return Una lista con las dos particiones
createTraintAndTestPartition <- function (dataset, SplitRatio = 0.75) {
  set.seed(123)
  dt <- list()
  sample = sample.split(dataset,SplitRatio = SplitRatio)
  dt[["train"]] = subset(dataset,sample ==TRUE)
  dt[["test"]] = subset(dataset, sample==FALSE)
  return (dt)
}

#'@function probCalc
#'
#'@param prop Resultados probabilísticos de los modelos
#'@param clases Número de clases que contiebe la variable de salida
#'@return Predicciones
probCalc <- function (prob, clases) {
  salida<-prob
  for(i in 2:(length(clases))) {
    salida[,i]<-prob[,(i-1)]*(1-prob[,i])
    salida[,1]<-(1-prob[,1])
    salida[,length(clases)]<-prob[,(length(clases)-1)]
  }
  return (salida)
```

```r
}

#'@function transformToBinary
#'@description Modificacion del datatset con valores binarios en la columna de salida por cada clase
#'
#'@param data data.frame de entrada
#'@param clase clase de la variable de salida
#'@return Dataset modificado
transformToBinary <- function(data, clase){
   num.column <- ncol(data)

  #creamos el dataset intermedio cambiando las clases en funcion del orden
  data[,num.column] <- ifelse(data[,num.column] > clase, 1, 0)
  data[,num.column] <- as.factor(data[,num.column])
  # Para cada conjunto de datos aprendemos un modelo
  colnames(data)[num.column] <- "target"
  return (data)
}

#'@function makePrediction
#'@description Generao un modelo en base al nombre pasado por parámetro y general la predicción
#'
#'@param clasificador Nombre del tipo de clasificador
#'@param trainData Dataset de train para generar el modelo
#'@param testData Dataset de test para hacer la prediccion
makePrediction <- function (clasificador, trainData, testData) {
  if (clasificador == "J48") {
    model <- J48(target~ ., data = trainData)
    return(predict(model, testData, type = "prob"))
  } else if (clasificador == "randomForest") {
    model <- randomForest(target~ ., data = trainData)
    return(predict(model, testData, type = "prob"))
  } else  {
    stop("Invalid clasificador")
  }
}

#'@function ordinalClassification
#'@description Aplicación de un modelo de clasificación de manera ordinal
#'
#'@param data data.frame de entrada. Se espera la variable de salida en la última columna
#'@return Lista de modelos.
ordinalClassification <- function (dataset, clasificador = "J48") {
  if(is.null(dataset)) stop("null dataset values not allowed")

  num.column <- ncol(dataset)

  data <- createTraintAndTestPartition(dataset)
  testData <- data[["test"]]
  trainData <- data[["train"]]

  #Obtenemos el número de clase del problema:
  clases<-as.integer(unique(dataset[,num.column]))
```

```
  #Creamos un vector del tamaño de test para contener las probabilidades
  prob <- 1:length(testData[,2])

  #Para cada clase menos la última
  for(i in 1:(length(clases)-1))
  {
    binaryTrainData <- transformToBinary(trainData, i)

    pred <- makePrediction(clasificador, binaryTrainData, testData)

    prob<-cbind(prob,as.data.frame(pred)$`1`)
  }

  salida <- probCalc(prob, clases)

  #Nos quedamos con el indice de la columna que tiene el mayor elemento
  pred = apply(salida[,-1],1,which.max)
  acc = sum(pred==testData[,num.column])/length(testData[,num.column])
  return(list("prediction" = pred, "accuracy" = acc))
}

pred <- ordinalClassification(esl, clasificador = "J48")
pred[["accuracy"]]
```

```
## [1] 0.6237113
```

```
pred <- ordinalClassification(era, clasificador = "randomForest")
pred[["accuracy"]]
```

```
## [1] 0.2625
```